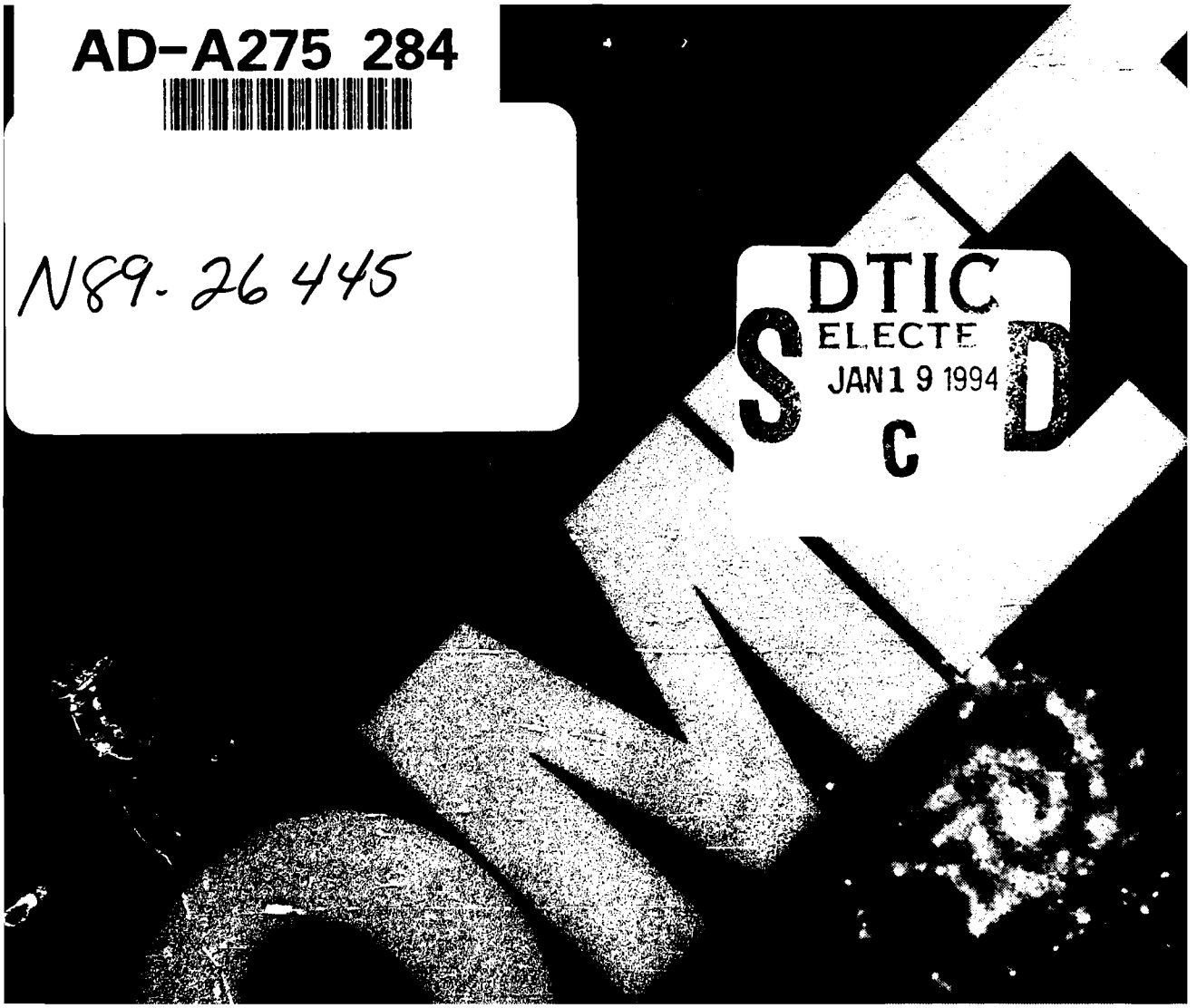

The NASA STI Program ... in Profile

Since its founding, NASA has been dedicated to ensuring U.S. leadership in aeronautics and space science. The NASA Scientific and Technical Information (STI) Program plays an important part in helping NASA maintain its leadership role.

The NASA STI Program provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program is also NASA's institutional mechanism for disseminating the results of its research and development activities.

A number of specialized services help round out the Program's diverse offerings, including creating custom thesauri, translating material to or from 34 foreign languages, building customized databases, organizing and publishing research results.

For more information about the NASA STI Program, you can:

- **Phone** the NASA Access Help Desk at (301) 621-0390
- **Fax** your question to NASA Access Help Desk at (301) 621-0134
- Send us your question via the **Internet** to help@sti.nasa.gov
- **Write to:**

  NASA Access Help Desk
  NASA Center for AeroSpace Information
  800 Elkridge Landing Road
  Linthicum Heights, MD 21090-2934

POLITECNICO DI MILANO

DIPARTIMENTO DI ELETTRONICA

RAPPORTO INTERNO

PIAZZA LEONARDO DA VINCI 32
20133 MILANO
ITALIA

NASA CENTER FOR AEROSPACE INFORMATION

800 ELKRIDGE LANDING ROAD LINTHICUM HEIGHTS, MD 21090 (301) 621-0390

ATTN: (PHONE)
DATE OF REQUEST: 01/07/94
USER ID: 02672
ORDER CONTROL NUMBER: 940107095746
USER SECURITY LEVEL: SECRET RESTRICTED

ENCLOSED IS YOUR ORDER FOR 0001 HARDCOPY COPY OF NASA ACCESSION NUMBER N89-26445
TITLE: GEKATOO: A general knowledge acquisition tool
REPORT NUMBER: REPT-89-033

DOCUMENT CLASSIFICATION: UNCLASSIFIED
DISTRIBUTION LIMITATION STATEMENT: UNLIMITED

QUESTIONS CONCERNING THIS ORDER SHOULD BE DIRECTED TO DOCUMENT REQUEST SERVICES, NASA CASI, (301) 621-0390.
PLEASE INCLUDE YOUR ORDER CONTROL NUMBER WITH YOUR INQUIRY.

02672-940107095746
DEPT OF DEFENSE
DEFENSE TECHNICAL INFORMATION CENTER
ATTN: DTIC-OCP/JOYCE CHIRAS
CAMERON STATION BLDG 5
ALEXANDRIA VA 22304

DIPARTIMENTO DI ELETTRONICA - POLITECNICO DI MILANO

GEKATOO: a general knowledge acquisition tool.

Andrea Bonarini,
Maria Caterina Gallo, Marco Guida.

Report n. 89-033

Marzo 1989

Title: GEKATOO: a general knowledge acquisition tool.

Authors: Andrea Bonarini, Maria Caterina Gallo, Marco Guida.

Report n.: 89-033

Abstract:

In this paper we outline the structure of a tool supporting a knowledge engineer (KE) in the knowledge acquisition process for building Knowledge Based Systems (KBS) for different kind of applications.

We believe it is impossible to make a machine performing automatically this process, since too many competences should be transferred from the KE to a computer. In order to develop an effective support system, a model of the KE's knowledge is needed.

GEKATOO implements a proposal for such a model, partitioning the expert's knowledge into three areas. The first two are related to the domain of application and the classes of application. The third one enables domain and application independent analysis of the reasoning methods to be used by the KBS to generate new knowledge from the one already present. The result of the application of GEKATOO is a conceptual model of the knowledge needed to implement a KBS solving the task defined by the expert.

A first prototype of the system has been implemented in Common Lisp and CLOS on Xerox 1186 machines.

Index terms: Artificial Intelligence, Knowledge Acquisition,.

--ooo0ooo--

To obtain a copy of this report please fill in your name and address and return this page to:

Laboratorio di Calcolatori
Dip.to di Elettronica - Politecnico di Milano
P.zza L. da Vinci 32 - 20133 Milano, Italy

NAME .....

ADDRESS .....

.....

Introduction

The implementation of a KBS involves different problems: knowledge acquisition (Clancey, 1985), knowledge formalization (Brachman 1979; Newell 1980; Breuker and Wielinga 1987), choice of the programming environment, and so on.

computer-based tool supporting the KBS development has to cope with all these problems at different levels.

We developed a conceptual framework (KRF - Knowledge Representation Framework) defining the different levels in terms of: conceptual modeling, representation formalism selection, formalization, and choice of an implementation environment (Bonarini, Gallo, Guida, in press).

According with Breuker and Wielinga (in press), we think that the true bottleneck of the development of a KBS concerns the expert's knowledge conceptualization more than its elicitation. In fact, the real problem is to make the conceptualization of the expert's knowledge fitting the reasoning mechanisms available to build KBSs.

In this paper we focus on the conceptual modeling process (Breuker and Wielinga, 1987) describing the corresponding part of KRF, namely GEKATOO (GEneral Knowledge Acquisition TOOl).

Our approach reflects the idea that a tool supporting the KE in this phases should allow him to follow the expert's own conceptual order in providing knowledge using the terminology he is familiar with.

The KE activity is supported by GEKATOO to achieve the syntactic completeness of the elicited knowledge, i.e. to ensure that all the entities mentioned are defined and related to other ones.

In order to support the KE in this activity, we partition his knowledge into three **areas**, corresponding to the different knowledge features needed to emulate the expert solving a problem.

The knowledge in the first area allows to describe the **domain of application** in terms of basic structures like **relation**, **element**, **state**, and so on.

The second knowledge area contains the description of the possible **classes of application** (such as diagnosis, planning, etc.) given by means of the basic **competences** needed to cope with the corresponding problems. Competences are defined in terms of **action schemata** at different level of abstraction.

In the last area, we define the characteristics of **reasoning methods** used to infer knowledge from given knowledge.

GEKATOO helps the KE, interacting with the expert, to generate the **conceptual model** from the structures given in the three areas.

The conceptual model is a formalization of the knowledge elicited from the expert to solve a particular problem.

This formalization is performed instantiating the structures represented in the three areas by means of primitives implemented in COMMON LISP and CLOS. This allows to "run" the conceptual model in order to test some of its features.

In the following, we will describe the parts of the system in details and show how it works on a simple application. In particular, we will show in details only the knowledge structures most relevant to understand GEKATOO. All the other knowledge structures are formally described in the GEKATOO reference manual (Bonarini, Cremonesi, Ferrari, Gallo, Guida, 1988).

The domain of application area

A domain representation is a collection of descriptions of **elements** and of **relations** among them.

The level of abstraction and complexity of the description depends on the specific domain and on the point of view of the expert. A domain element can be a simple object or a class of objects. Therefore, a relation can hold among objects, classes or combinations of the two.

The KE activity is performed according to his own knowledge acquisition strategies. On the other side, the Expert, which owns the knowledge about the problem, should be allowed to provide it according to his own model. The interaction between the two is supported by GEKATOO in order to enable a cooperative Knowledge Acquisition process.

Thus, the interaction with the expert may start defining a relation or an element, at his choice, since both the alternatives can be accepted by the KE.

In this knowledge area we define a structure for the description of domain elements:

ELEMENT
NAME: element_name
IS_A: class_name
ATTRIBUTES: element_attr_list
STATES: state_list

The slot **NAME** contains a unique identifier for the element to be defined.

The attribute IS_A contains the name of the class which the element belongs to.

The slot ATTRIBUTES is filled with the list of the attributes of the element considered relevant by the domain expert. Each attribute is fully defined by a structure describing: whether its value can vary in time or not, its description unit (e.g. number, kilometers, color, ..) and its acquisition method (see below).

The slot STATES contains the list of the states relevant for the element. Each state is described by a name and a list of constraints on attributes values.

Each relation is described according to the following structure:

RELATION
NAME: relation_name
PARTICIPANTS: arguments_list
TYPE: relation_type
ACQUISITION_METHODS: a_m_list
ASPECTS: aspects_list

The slot **PARTICIPANTS** contains the ordered list of the entities involved in the relation. An **entity** may be an object, a class of objects, an attribute or a state.

For instance, the relation PART_OF_201 may hold between the class ENGINE and the class CAR, while the relation PART_OF_90 may hold between the object FIRE_ENGINE_#1249947 and the object MY_FIAT_TIPO.

The content of the slot **TYPE** should be a predefined reference name, identifying a class of relations. It allows to refer to relations more efficiently. This is used by knowledge structures belonging to the classes of application area, and will be further discussed in the next section. In order to fix ideas now, let us think at TYPE names such as: CAUSE-EFFECT, MATHEMATICS, etc. For instance, the relation GREATER_27 is a MATHEMATICS relation, while the CAUSE_POISONING_908 one is a CAUSE_EFFECT relation.

The slot **ACQUISITION_METHODS** can be used to record means to verify if the relation holds in the actual world. The contents of this slot will be further discussed in the following.

The slot **ASPECTS** can be used to add information at deeper levels. For instance it can be used to describe the functional, causal, temporal, or teleological aspects of the relation. The description of each aspect needs a different structure.

Acquisition methods can be associated both to relations and to attributes of elements, in order to define how they can be obtained either from the real world or by inference from already acquired knowledge. In the following we will describe the acquisition methods structure for the attributes, being the relations ones analogous.

Each acquisition method can be described using the following structure:

ACQUISITION_METHOD
NAME: a_m_name
TYPE: a_m_type
SOURCES: info_sources
DESCRIPTION: a_m_description
APPLICABILITY_COND: state_list
COST: a_m_cost

The slot TYPE states whether the acquisition method concerns **deduction** or **observation**. In the first case the attributes which the method refers to are deduced from relations and attributes. In the second case, the attribute values are obtained from the world, through interaction with the user. In both cases, the domain expert must decide which acquisition method is more adequate and supply the knowledge needed to describe it.

The slot SOURCES defines the sources of information which the acquisition method will be applied to. Each source is described in terms of single elements or pairs (element . attribute).

The slot DESCRIPTION contains a description of how the acquisition method should be applied. In the case of "observation" the slot is filled with the acquisition procedure, defined as a set of instructions for the end user. In the case of "deduction", the slot contains the keyword INFER, meaning that a generic inferential mechanism should be applied to the SOURCES defined by the domain expert in order to obtain the required information.

The slot APPLICABILITY_COND contains the list of the states to be verified in order to actually apply the acquisition method.

The attribute COST defines the cost of the application of the method, as stated by the domain expert on a given numerical range.

The classes of application area

The classes of application are inspired by the concept of category of application presented by many authors (Kitto and Boose, 1987; Clancey, 1986; Hayes-Roth et al., 1983).

In this knowledge area we define a class of application in terms of the **competences** underlying the execution of the related tasks. We define a competence as a set of specific actions together with the abstract entities on which to perform them.

An **abstract entity** is a place holder specific for each class of application and referring to entities described in the domain of application. The interaction with the expert is aimed at relating these abstract entities to domain structures.

The use of abstract entities allows to define a class of application independently from the domain description.

We assume that the competences are partly shared among different classes of application, and partly specific to each of them.

In GEKATOO, competences are described as **action schemata.**

An action schemata is defined using the following structure:

ACTION
NAME: action_name
ELEMS: elem_type_list
MODALITY: plan

The slot **ELEMS** contains the list of the abstract entities characteristic for the action. A name is given to each abstract entity in order to refer to it within the description of the action. Moreover, to each name is associated the type of the domain entity which the abstract entity refers to. For instance, the ELEMS list for the action Recover is ((Malfunctioning . STATE)).

In a similar context, Breuker and Wielinga (in press) introduce the concept of metaclasses, in order to represent semantic knowledge about the elements involved in their task description layer (analogous to our classes of application area). In GEKATOO, all the semantic knowledge associated to the ELEMS concerns their use and it is embedded in the MODALITY description of the ACTION.

The slot **MODALITY** describes the plan for the execution of the action. It is defined organizing actions at a lower level of abstraction by means of the canonical control structures: sequence, condition, and while-loop. Each action referred to in the plan description is, in turn, described by an action schema. The plan description language we use is implemented using COMMON LISP in order to obtain an executable action specification.

Some of the actions MODALITY can refer directly to a class of reasoning methods. The representation of such methods is described in the reasoning method area.

The selection within the specified class of a suitable reasoning method for a specific action is up to the KE, which has to perform this task stimulating requirements from the expert.

The Reasoning Methods area

In this area are described the methods which can be used to deduce new knowledge from already acquired knowledge.

The kind of knowledge belonging to this area is different from the one of the other areas. In fact, here we give prototypical descriptions of deduction procedures to be used by the KE, in the form supplied in GEKATOO. The only interaction with the expert needed by this area is aimed to identify the domain entities to be used in such procedures.

The **reasoning methods** (e.g. "plain backward chaining", "CF backward chaining", "generate and test") are defined in terms of operations to be performed on **reasoning entities**. A reasoning entity structure (e.g. "p-rule", "frame") is described by means of a set of components. For instance, the production rule can be defined as:

```
REASONING ENTITY
   NAME: P-RULE
   COMPONENTS: ((Premise . (LIST_OF DOMAIN_ENTITY))
                (Consequent . (LIST_OF DOMAIN_ENTITY)))
```

Reasoning methods are described giving the different contents of the following slots:

- NAME: a unique identifier for the reasoning method.
- TYPE: one of the predefined general types of reasoning methods. This types are identified on the basis of an operating strategy common to a set of reasoning methods. For instance the BACKWARD- CHAINING type is used for SIMPLE-BACKWARD-CHAINING, BACKWARD-CHAINING-ON-FRAMES, and CF-BACKWARD-CHAINING.

- REASONING_ENTITIES: a list of reasoning entities which the reasoning method algorithm refers to.
- CHAR-ELEM: list of the variables referred to in the algorithm. Each of them is associated with the type of domain or abstract entity on which the algorithm has to work.
- ALGORITHM: the description, given in an executable language, of the algorithm used by the specific reasoning method. This description may contain also explicit references to different reasoning methods. For instance, the BW-FW reasoning method algorithm refers to SIMPLE-BACKWARD-CHAINING and SIMPLE-FORWARD- CHAINING, making explicit their connections step by step.

An example of the use of GEKATOO

Let us have a look to the interaction between a KE and an expert aimed at the conceptualization of the knowledge required to diagnose car electrical malfunctioning.

Let us suppose that part of the conceptual model has been already defined. Namely, we already know that the car has an electrical plant, which is constituted by some devices like: a battery, a starting gear, headlights, starting connections, and cables connecting all of them.

In this case, the expert shown a preference for the description of domain elements and relations among them, instead of focusing on diagnosis procedures. In particular, in the conceptual model, we have an instance describing a battery in terms of:

ELEMENT
NAME: Battery
IS_A: Electrical_device
ATTRIBUTES: ()
STATES: (Dead)

Moreover, let us suppose that the following relation had been just defined:

RELATION
NAME: Power_003
PARTICIPANTS: ((Battery . Working) (Headlights . Working))
TYPE: CAUSE_EFFECT
ACQUISITION_METHODS: ()
ASPECTS: ()

This definition is incomplete, since the Working state of Battery and the structure describing Headlights have not been yet defined. Therefore, according with the principle of syntactic completeness, which rules the conceptual modeling process, those entities are put in waiting lists for undefined states and undefined elements.

At this time, many undefined entities should be specified. The choice of the next entity to be defined is performed by the KE on the basis of his own experience and intuition. In this case, the GEKATOO interface provides a set of active windows supporting the acquisition of the undefined state.

The need to define "Working" leads to the specification of a constraint stating that the Voltage of the battery should be GREATER than 12 Volts. Since the attribute Voltage had not been defined, it enters in the undefined attributes list.

This process may continue in the same way, until the expert considers that all the relevant entities involved in the task execution are described. At this point, the KE may try to define the competences needed to diagnose that kind of malfunctioning.

The classes of application area provides action schemata describing those competences.

Let us see now one of the most relevant action schemata provided for this class of application.

```
ACTION
   NAME: Reconstruct_causes
   ELEMS: ((Effect_to_be_justified . (STATE))
           (Selected_cause . (STATE)))
   MODALITY:
      (LET ((Causes
             (deduce
              (Reasoning_method (TYPE BACKWARD_CHAIN))
              (Relation (TYPE CAUSE_EFFECT))
              Effect_to_be_justified)))
         (IF ( 1 (LENGTH Causes)
             (SETQ Selected_cause (select Causes))  ;then
             (SETQ Selected_cause (FIRST Causes)))) ;else
```

The MODALITY specifies that the action consists in finding for a given effect the relationships with its possible causes as defined by the corresponding structure in the domain.

This action schema instance states that the search should be performed on relations of type CAUSE_EFFECT and using backward chaining in order to identify causal chains. If more than one cause is found, a selection must be performed. The competence referring to "select" is detailed by another action schema instance.

This description of how we would like the KBS work, is used at a meta-level in order to state the need for the definition of all the possible causal chains for every Effect, and the classification of the domain states considered Effects by the expert.

The connection with the reasoning methods area is due to the invocation of the action "deduce".

The type of reasoning method is predefined in the action schema, and it is used to limit the choice among the possible reasoning methods to be applied in this class of application.

In GEKATOO the type of reasoning method to be used is predefined, according to considerations about the characteristic elements of the class of application and the type of domain entities which they refer to.

When the slot MODALITY contains the action "deduce", the KE has to map the reasoning entities with the characteristic elements of the current action and the type of domain entities which the elements themselves refer to.

In our example, the KE may choose the SIMPLE_BACKWARD_CHAINING reasoning method, which works on P_RULES. Thus, given the current situation, he has to elicit the chain of relations of TYPE CAUSE_EFFECT which may be backward_chained in order to deduce the Cause from the Effect.

Discussion

The state of the art in knowledge acquisition models reveals that research efforts have been mainly concerned with modeling classes of application. We propose a complete model of the knowledge involved in the conceptualization process. In fact, GEKATOO embeds structures to represent knowledge about classes of application, domain, and reasoning methods, and the relationships among them.

GEKATOO is conceived as an interactive tool for the KE, supporting his interaction with the expert, but leaving him the control of the knowledge acquisition process as a whole.

At present, GEKATOO does not account for the problems underlying semantic completeness of the conceptual model. In particular, as regard the elicited knowledge, it doesn't consider issues of homonymy, sinonymy, inconsistency and lack of a whole piece of knowledge. This last problem regards cases in which part of the not yet elicited knowledge is loosely connected with the one already defined.

References

- Bonarini A., Cremonesi C., Ferrari A., Gallo C., Guida M., 1988, **GEKATOO: the user manual**, Milan Polytechnic Artificial Intelligence Project, Department of Electronics of Politecnico di Milano.
- Bonarini A., Gallo C., Guida M., in press, KRF: a methodological framework for representing knowledge, **Computers and Artificial Intelligence**, 29-2.

- BrachmanR.J., 1979, On the epistemological status of semantic networks. In N.V.Findler (ed.) **Associative networks: representation and use of knowledge by computers**, Academic Press, New York, 3-50.
- Breuker J., Wielinga B., 1987, Use of models in the interpretation of verbal data. A. Kidd (Ed.) **Knowledge elicitation for expert systems: a practical handbook**, Plenum Press, New York.
- Breuker J., Wielinga B., in press, Models of expertise in knowledge acquisition, in Guida G., Tasso C. (Eds.), **Expert Systems Design**, North-Holland.
- Clancey W.J., 1985, Heuristic Classification, **Artificial Intelligence**, 27 215-251.
- Hayes-Roth F.,Waterman D.A., LenatD.B. (eds.), 1983, **Building Expert Systems**, Addison-Wesley Publ. Company, Readings, MA.
- Kitto C.M., Boose J.H., 1987, Selecting knowledge acquisition tools and strategies based on application characteristics, **Proc. 2nd AAAI Knowledge Acquisition for KBS Workshop**, Banff, Canada.
- Newell A., 1980, A physical symbol system. **Cognitive Science**, 4, 135-183.